MOSQUITO 1.5
User Manual

SEAlab - Software Quality Group
Dipartimento di Informatica
Università dell'Aquila

May 2008

1 MOSQUITO VERSIONING 3

2 GETTING STARTED 4

2.1 Installation Requirements 4

2.2 Getting Eclipse with Eclipse UML 4

2.3 Installing MOSQUITO 5

3 MOSQUITO AT A GLANCE 6

4 THE SAP·ONE METHODOLOGY 10

5 THE PRIMA-UML METHODOLOGY 10

6 MOSQUITO AT WORK 11

6.1 eCommerce System Specification 11

6.2 Use Case Diagram 12
6.2.1 Sap·One annotations 13
6.2.2 Prima-UML annotations 13

6.3 Component Diagram 14
6.3.1 Sap·One annotations 16
6.3.2 Prima-UML annotations 19

6.4 Sequence Diagram 19
6.4.1 Sap·One annotations 22
6.4.2 Prima-UML annotations 24

6.5 Deployment Diagram 25
6.5.1 Sap•One annotations 27
6.5.2 Prima-UML annotations 27

7 REFERENCES 31

8 APPENDIX 32

8.1 Queuing Network process creation (SAP·one methodology) 32

8.2 Execution Graph creation (Prima-UML methodology) 32

1 Mosquito versioning

The MOSQUITO 1.5 release provides new features that have been added to the previous version. In particular now MOSQUITO can be invoked with different exports formats and a new functionality to completely execute the PRIMA-UML methodology has been added.
The MOSQUITO evolution across the different releases is exposed in the picture here below.

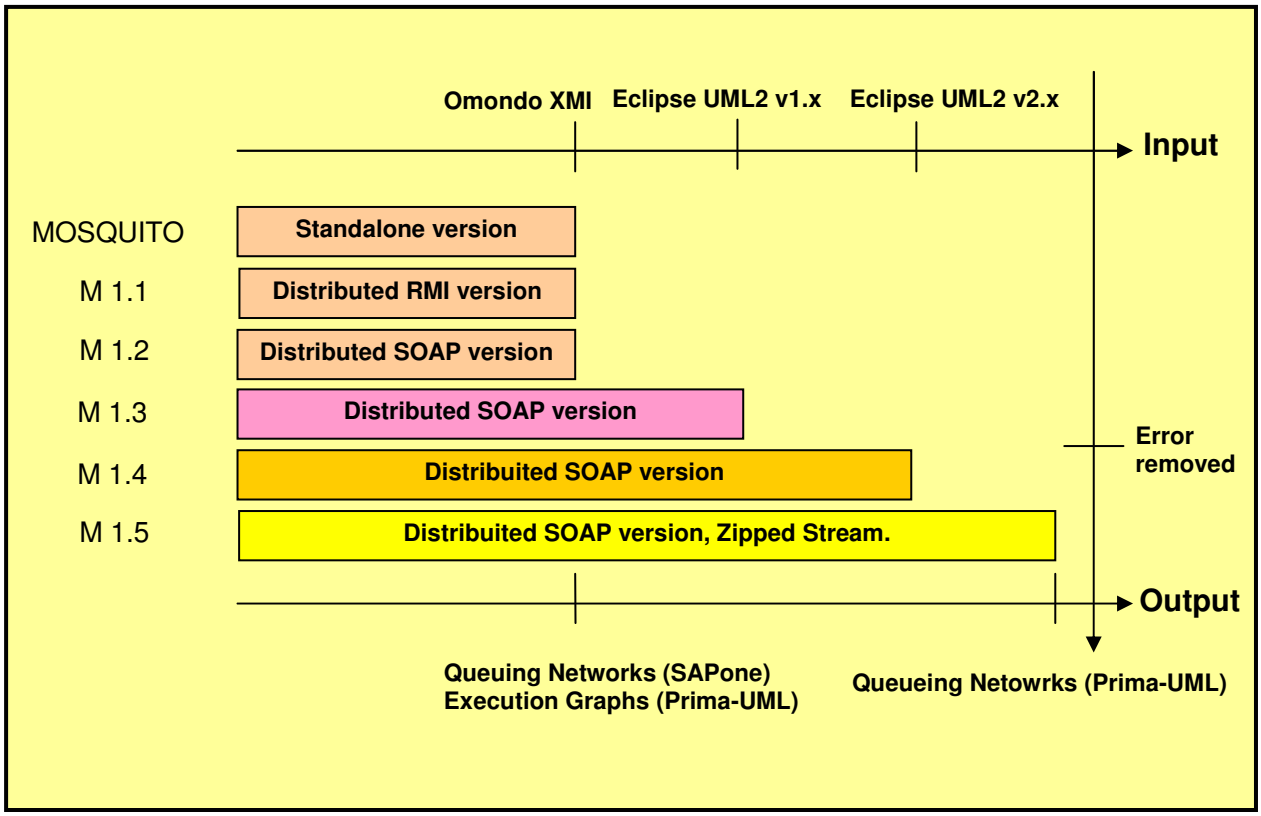


Figure 1: Mosquito versioning.

The last MOSQUITO release (i.e. M 1.5) is able to build, besides Execution Graphs and QN at the software level, also QN at the platform level that can be combined with Execution Graphs for detailed parameterization.
The MOSQUITO input has been extended to include an Eclipse UML v2 annotated Deployment diagram, in agreement with the requirements of Prima-UML methodology.
The current release of MOSQUITO provides a log error feedback that is very useful to locate a problem in the input model.
The following picture show all the use cases that MOSQUITO 1.5 provides to the users.

2 Getting Started

MOSQUITO (MOdel driven conStruction of QUeuIng neTwOrks) is a tool developed in JAVA language as plugin for the Eclipse Platform. To use it you need to install a Java Virtual Machine and the Eclipse Platform with Eclipse UML.

2.1 Installation Requirements

Server Side

Operating System:	Independent
Web Application Container:	Apache Tomcat 6.0 with Axis2 1.4
JDK version:	1.5 or higher

Client Side

Eclipse:	3.2 or higher
Eclipse EMF:	2.2.x
Eclipse UML2:	2.x
JDK version:	1.5 or higher

2.2 Getting Eclipse with Eclipse UML

The main Web site for Eclipse is www.eclipse.org. On that page, you can see the latest Eclipse news and links to a variety of online resources, including articles, newsgroups, bug tracking, and mailing lists.
The latest version of Eclipse can be downloaded from the main download page at www.eclipse.org/downloads (if this site is unavailable, a number of mirror sites around the world are available). Go ahead and download the latest release or stable build.
The download page for each release includes various notes concerning that release as well as links to every platform version. Eclipse supports a large number of platforms, including Windows, Linux, Solaris, HP, Mac OSX, and others. Choose the Eclipse SDK download link corresponding to your platform and save the Eclipse zip file to your computer's hard drive. This will generally be a very large file (>105 MB), so be patient unless you have sufficient bandwidth available to quickly download the file.
Once the Eclipse zip file has been successfully downloaded, unzip it to your hard drive. Eclipse does not modify the Windows registry, so it does not matter where it is installed.
MOSQUITO was developed with Eclipse 3.2 and subsequently tested with Eclipse 3.2 and higher. It is very important to download a version of Eclipse with the Eclipse UML plugin already installed, to create the UML models that represent the input to MOSQUITO. However, if you already have an Eclipse installation without Eclipse UML plugin, you can update your platform with the update manager selecting Help **-->** Software Updates **-->** Find and Install **-->** Search for New features to Install.

2.3 Installing MOSQUITO

You can download MOSQUITO client from the following web page
http://sealabTools.di.univaq.it/SeaLab/MosquitoDownload.html.

MOSQUITO is a plugin of the Eclipse platform, and its installation (like every other Eclipse plugin) is very simple. It is sufficient to unzip the zip file di.univaq.MOSQUITO in the main folder of Eclipse (e.g., c:/eclipse). After this step verify if the plugin jar file appears in the plugin directory (e.g., c:/eclipse/plugins/di.univaq.MOSQUITO_1.5.0.jar) and you must be sure that previously installed MOSQUITO versions have been removed from the system.
Eclipse caches plug-in information in a configuration directory. For this reason, the first launching of Eclipse with the installed MOSQUITO plugin must be performed using the –clean command-line option so that the cached plugin-in information will be rebuilt.
To do this you must use the prompt to launch Eclipse. If you are in the directory of Eclipse installation them you must type the "Eclipse –clean" command as reported in picture here below.

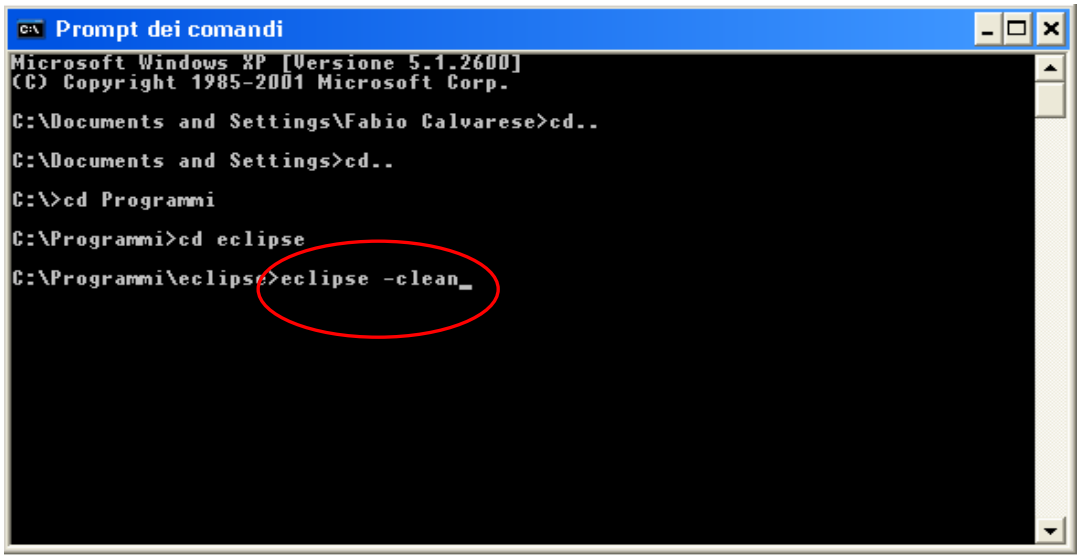


Figure 2 Eclipse refresh procedure

After started the Eclipse platform, you can verify the correct MOSQUITO installation by inspecting the plug-in details under Help -> About Eclipse SDK -> Plug-in Detail. The MOSQUITO plugin must appear in the list of the currently installed plugins.

3 MOSQUITO at a glance

MOSQUITO (MOdel driven conStruction of QUeuIng neTwOrks) is a tool that generates Queueing Network models from UML software models according to the SAP·one [1] and the Prima-UML [2] methodologies.
MOSQUITO allows to annotate UML software models with performance data and finally creates an XML representation of performance models. MOSQUITO in its current release can create three different models, that are:
(i) Queuing Networks based on the SAP·one methodology;
(ii) Execution Graphs based on the first part of the Prima-UML methodology;
(iii) Queueing Networks based on the Prima-UML methodology.

MOSQUITO has a client/server architecture. The plugin di.univaq.MOSQUITO implements the client side that provides the functionality to invoke the web services placed at the University of L'Aquila. The system architecture is shown here below.

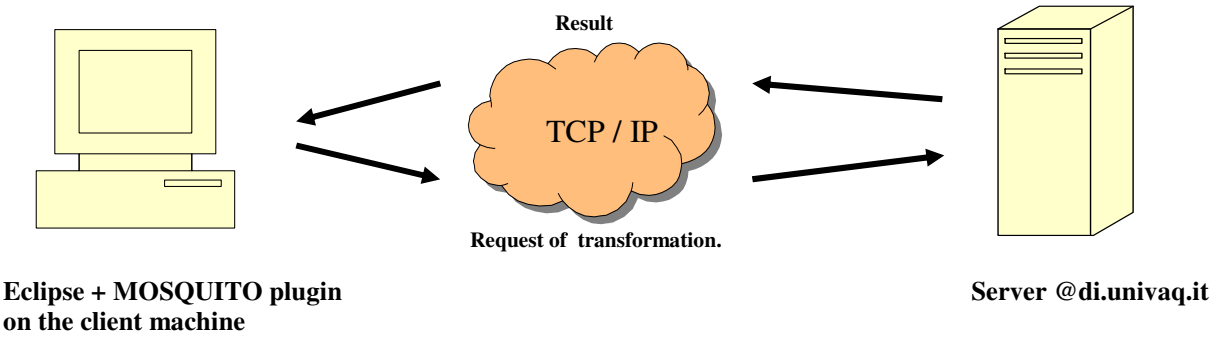


Figure 3: Mosquito client-server architecture.

In particular MOSQUITO is exposed as a web service. This will allow (in next MOSQUITO releases) any user to implement an alternative client, using the preferred languages and technologies, to use the services of MOSQUITO server placed at di.univaq.it domain.The figure here below shows the flow of requests (with the Soap protocol) to the Axis's APIs that are able to expose the registered services at high level.

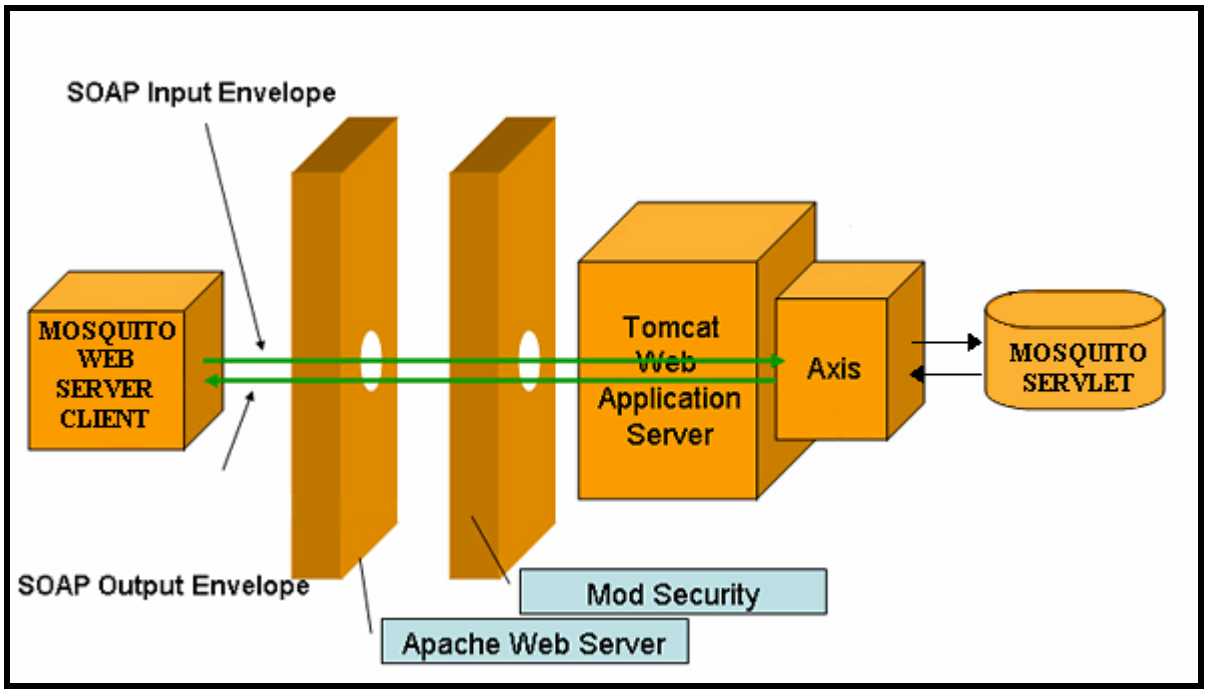


Figure 4: Mosquito Web Service architecture.

The MOSQUITO client that we provide is an Eclipse Plugin that allows an entirely automated process. The user must only properly create and annotate UML models according to the selected methodology (i.e SAP·one or Prima-UML).
The extension point used for the MOSQUITO interface is the Eclipse popup menu. After the creation of a model the user must only select the UML files (*.uml) and activate the desired transformation, as shown in the following figure.

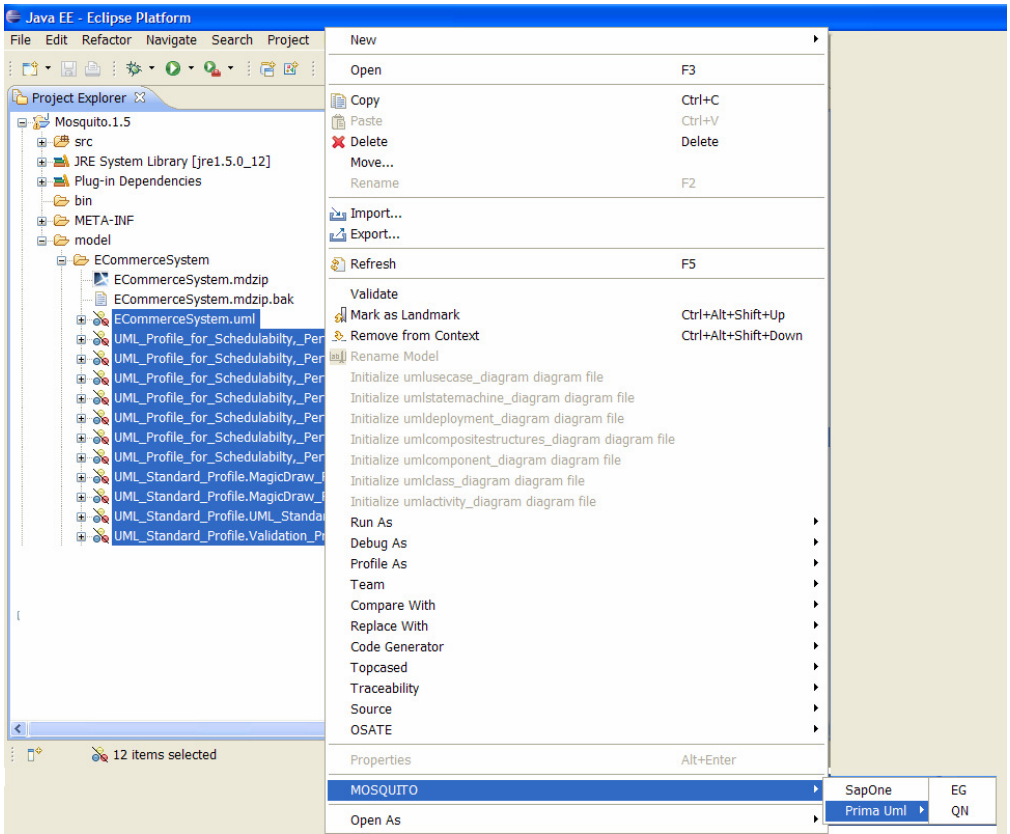


Figure 5: Invoking Mosquito transformations from Eclipse Platform.

The result of the transformation will be showed in the Eclipse XML editor as follows: eventually, a WARNING or ERROR tabs will be displayed containing the warnings or errors about input model, respectively.

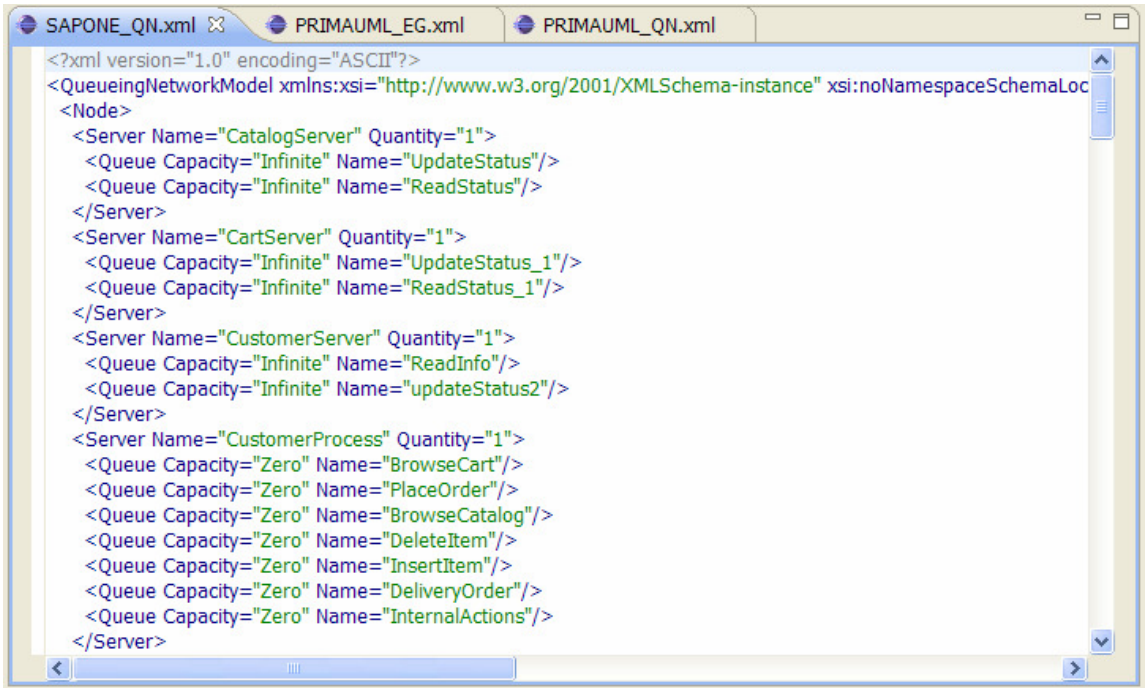


Figure 6: Mosquito output model: xml-based notation for SapOne QN model.

The picture below shows the use of MOSQUITO tool in a performance analysis process. The light orange shaded area contains the step under development.
The entire process starts with the construction of an UML model using any UML tool exporting in Eclipse UML 2.
The two methodologies available in MOSQUITO (i.e. SAP·one and Prima-UML) use different sets of diagrams to model the software system.
Along the SAPone methodology path a QN model is created that represents the system at the software architectural level and can be solved to obtain performance measures at the software level.
Along the Prima-UML methodology path two intermediate models are created (i.e. an EG and a QN) that need to be merged to produce a parameterized QN. The latter can be solved to obtain performance measures keeping into account the software and the hardware characteristics [4].

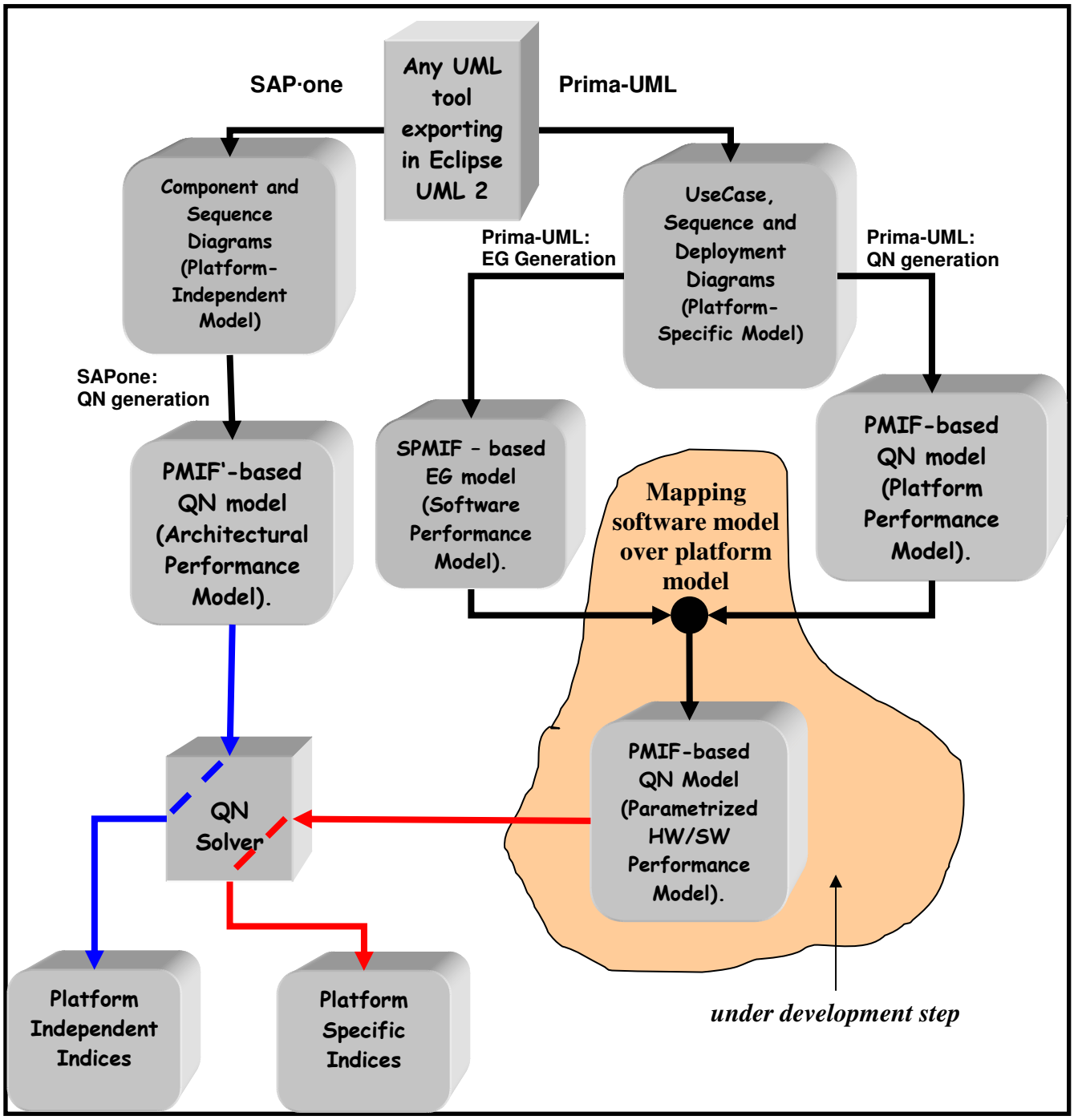


Figure 7: The performance analysis process carried out by Mosquito.

4 The Sap·One methodology

SAP·one [1] is a methodology for the performance modelling of a software system. It consists essentially in the annotated generation of a queuing network from an UML architectural model.
SAP·one has been conceived to be applied in the first phases of the software process to reveal architectural flaws as soon as possible, so the developer will be able to modify the system architecture.
The UML model is enriched with additional information regarding the system performance, such as scheduling policies and service times of software components.
In this approach the service centers of the queuing network don't represent hardware resources (such as disks or processors) rather they represent the software components of the system architecture.
The UML model represents therefore a platform-independent model, and the performance indices that can be obtained are "somehow" parametric. This is because the assumption underlying the methodology is that every software component will be placed on a logical device, and that all the devices will have the same "speed" but they can manage requests through queues of different capacity and different scheduling policy
In practice, the SAP·one analysis is comparative, in that the obtained performance values are useful to reveal bad architectural choices, but they are of little value as absolute performance indices.

5 The Prima-UML methodology

Prima-UML [2] is a methodology for the performance modelling of a hw/sw model. This methodology allows the generation from UML of two models, that are:

(i) a software performance model represented as Execution Graph (EG)
(ii) a platform performance model represented as an Extended Queuing Network Model (EQNM).

The merging of the EG and the EQNM models gives rise to a parameterized EQNM that ends the model generation phase. This type of system analysis is better suited when the developers have already some ideas on the potential target platforms, in that QN server centers in Prima-UML represent hw resources.

The Prima-UML methodology, starting from properly annotated UML2 diagrams, generates a sw/hw performance model. In particular an Use Case Diagram is used to obtain the system usage profile, the Sequence Diagrams for the Execution Graph generation and the Deployment Diagram for the EQNM topology generation.

6 MOSQUITO at work

In this section we explain how to introduce the SAP·One and Prima-UML annotations in the Eclipse UML models[1]. The presentation is driven by the eCommerce System example.

6.1 eCommerce System Specification

In the electronic commerce system, there is a supplier that publishes his catalogue on the web. The catalogue can be visioned by registered and unregistered users. Registered users become customers for the supplier. The supplier accepts customer orders and delivers the ordered items maintaining all the relevant data. He needs to maintain information on data, on the catalogue and on the orders purchased by his customers. Each registered customer has a cart where he/she can insert or delete items. The customer can order only if the cart is not empty. The system also allows the customer to monitor the order status and to confirm the delivery in order to permit the payment.

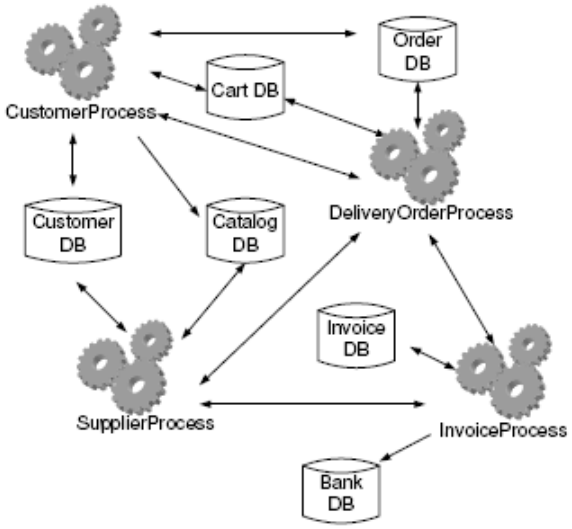


Figure 8: Overview of the eCommerce System

Figure 8 shows the system structure at a high level of details. In a first analysis, some databases (Customer DB, Cart DB, Order DB,etc.) and four process type (CustomerProcess, SupplierProcess, etc.) are identified. For each involved database, there is a server that permits to communicate with it. The interactions with these servers are asynchronous. Each customer has associated a (individual) CustomerProcess. If the customer is not registered, the process allows the browsing of the catalogue only, whereas if he is a registered one, the process activates all the functionalities the system provides him (cart managing, order placement, etc.). All the functionalities provided by the system to the supplier, instead, are coded in the SupplierProcess. The DeliveryOrderProcess and InvoiceProcess are processes that manage the activities to be executed to dispatch an order and to produce the corresponding invoice. In the system one DeliveryOrderProcess instance and one Invoice Process instance are running, any time, for each order and for each invoice in process, respectively.

[1] Generating an Eclipse UML2 model (.uml) as input to Mosquito implies a more detailed description using the *terms* used by the OMG UML 2.1 Superstructure specification (e.g the names used for metaclasses and their tags) and by an UML2 SPT profile (e.g. names used for stereotypes and their tags). That kind of detailed and precise should allow a tighter mapping between the Import/Export API code and its textual description: Eclipse UML uses the official OMG UML 2.1 Superstructure specification to define its own API.

For the sake of presentation, a simplified version of this system is considered. The focus is on a subset of customer functionalities. In particular, only catalogue browsing, cart browsing, item insertion and deletion to/from the cart, order placement functionalities will be considered.

6.2 Use Case Diagram

Figure 9 shows the use case diagram (UCD) for the e-commerce system. Three actors are in the system: Customer, Supplier and Bank. The e-commerce customer can browse both the cart (BrowseCart) and the catalogue (BrowseCatalog), she/he can insert and delete items to/from the cart (InsertItem and DeleteItem, respectively). She/He can place an order and check its status (PlaceOrder and BrowseOrderStatus), and finally she/he can confirm the delivery (ConfirmDelivery) of the items ordered. The use case diagram also specifies that both InsertItem and DeleteItem functionalities use the BrowseCart in their execution.

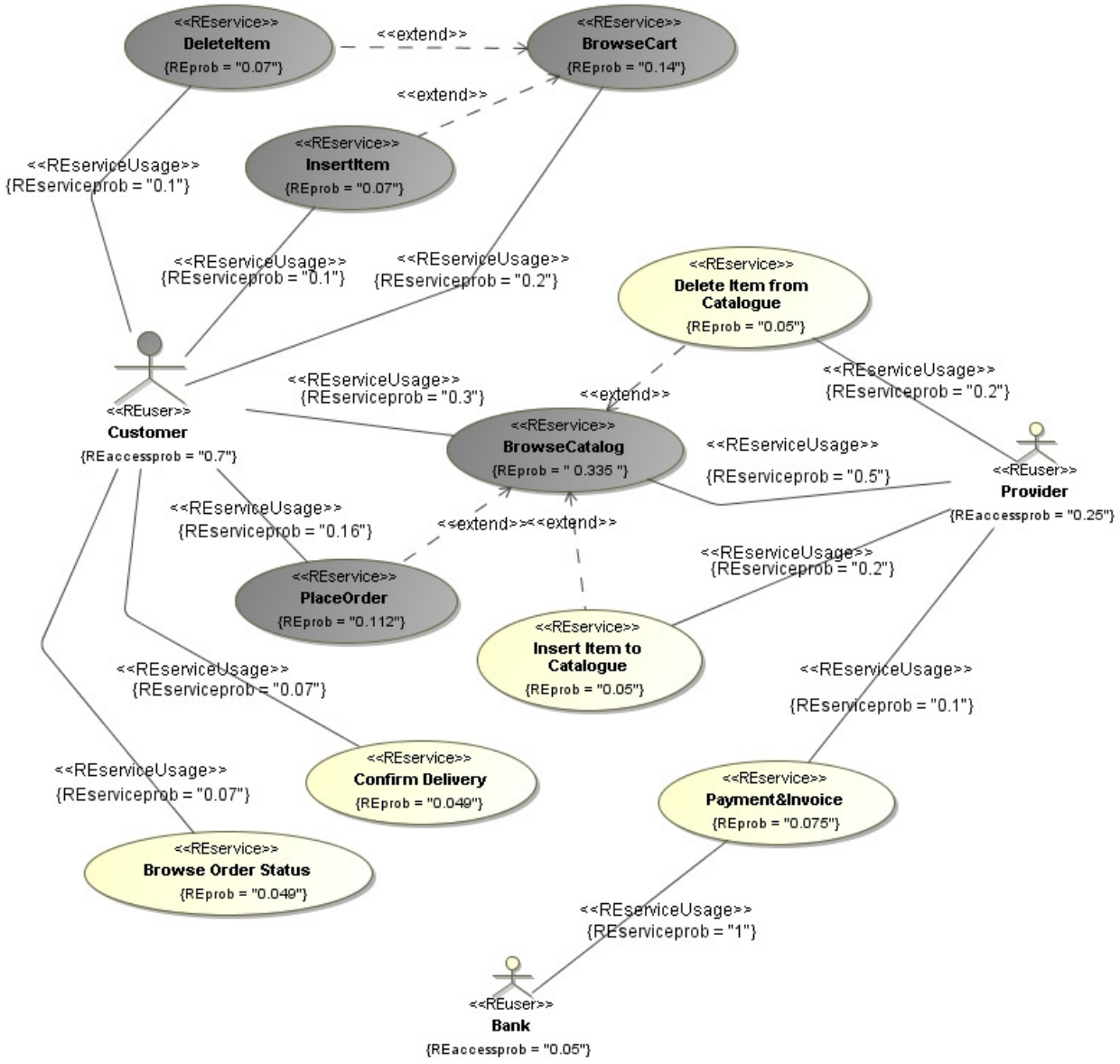


Figure 9: UCD of the eCommerce System with the operational profile.

The supplier maintains the catalogue by inserting new items from the catalogue (InsertNewItemInCatalogue) and deleting from it the items no more available

(DeleteItemFromCatalogue). Moreover, she/he processes the orders by delivering the items sold (DeliveryOrder) and producing the invoice after the customer has payed (Payment&Invoice). In this last use case, the bank actor is in charge for the money transfer from the customer bank account to the supplier one. Again, the use case diagram specifies that both InsertNewItemInCatalogue and DeleteItemFromCatalogue functionalities use the BrowseCatalogue in their execution.
For the sake of presentation, in this document a simplified version of this system is considered. The focus is on the customer view by considering only the software system functionalities reported on the figure. The choice has been driven by the fact that the focus is on performance aspects hence the customers are the system actors producing more workload to the system. Thus, the use cases critical from performance perspective are those accessed from them.

6.2.1 Sap•One annotations

No Sap•One annotations are required on the Use Case Diagram.

6.2.2 Prima-UML annotations

Prima-UML annotates and makes use of the Use Case Diagram (UCD) to take into account the operational profile.
The UCD must be annotated with information regarding the probability of a user to access the system, as well as the probabilities to invoke different use cases.
This information is introduced with the **stereotype «REuser»**[2] associated to an Actor and on the Actor-Use Case association, respectively, through the *ReAccessProb* and *ReService* tags.
The probabilities of the Use Case Diagram will be used to partially parameterize the performance model.

Stereotype Syntax	Apply Stereotype on (UML metaclasses)	Semantic
<<REuser>> {REaccessprob = value}	On the uml.**Actor**	It denotes the probability that an user accesses the system
<<REservice>> {REprob = value}	On the uml.**UseCase**	It denotes the probability that a provided system functionality can be accessed by all possible users.
<<REserviceUsage>> {REserviceprob = value}	On the uml.**Association** between **<<REuser>>** and **<<REservice>>**	It denotes the probability for the user to invoke a particular use case.

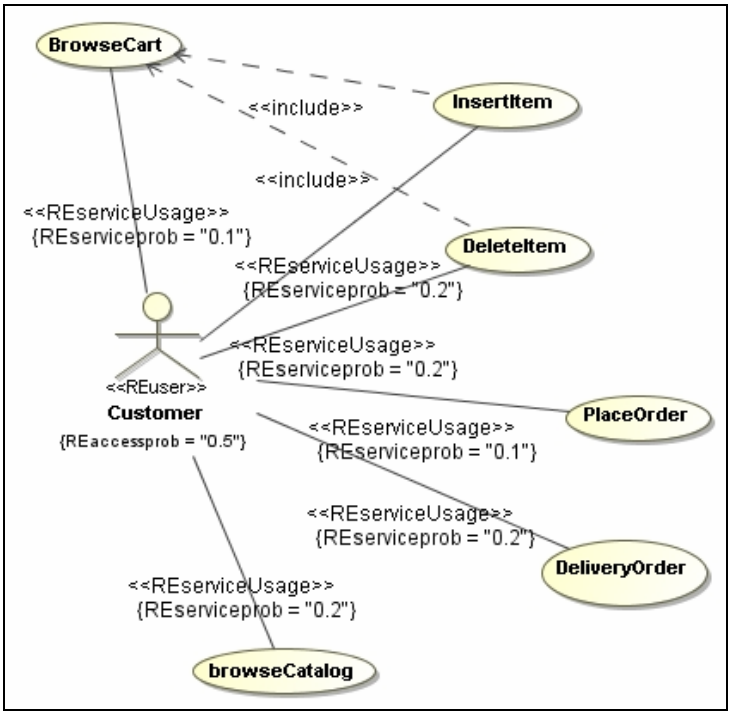
Table 1: Prima-UML Stereotypes used on UCD.

6.2.2.1 Assumptions

1) Each actor must to be stereotyped with the <<REaccessprob>> stereotype that allows to specify the probability of the actor to access at the system functionality. This value must fall between zero and one.

[2] UML SPT profile doesn't provide stereotypes to model the operational profile on the Use Case Diagram. We extend UML SPT with the <<REuser>> stereotype. Its name comes from another profile introduced for reliability purposes.

2) Each (actor - use case) connection must be stereotyped with the <<REserviceUsage>> stereotype that indicates the probability to access to the related use case. This value must fall between zero and one, as shown in the following figure.



6.3 Component Diagram

In Figure 10 we present the UML 2.0 Component Diagram (CD) for the portion of the e-commerce system we consider. This diagram highlights the software components and their required and provided interfaces. The provided interfaces are represented by a circle whereas the required ones are represented by a semicircle.
In general, an interface is composed by a set of operations provided or required by a software component. Here, to simplify the presentation of the parameterization phase, we assume that an uml.Interface represents uml.Operation of the uml.Component. Then each uml.Operation is annotated, according to the UML SPT profile[3], in order to introduce performance aspects of software components. All the values for the execution times of the uml.Operation are assumed (see the *assm* keyword in the tag value), as our intent is to carry on a performance analysis at the software architecture level when such information is not available, whereas it can be guessed by the designers on the basis of their experience or on the basis of previous releases of the same software system.

[3] Note that we *extend* the <<PAstep>> stereotype with the *Operation metaclass* in order to allow the annotation of the component Operation.

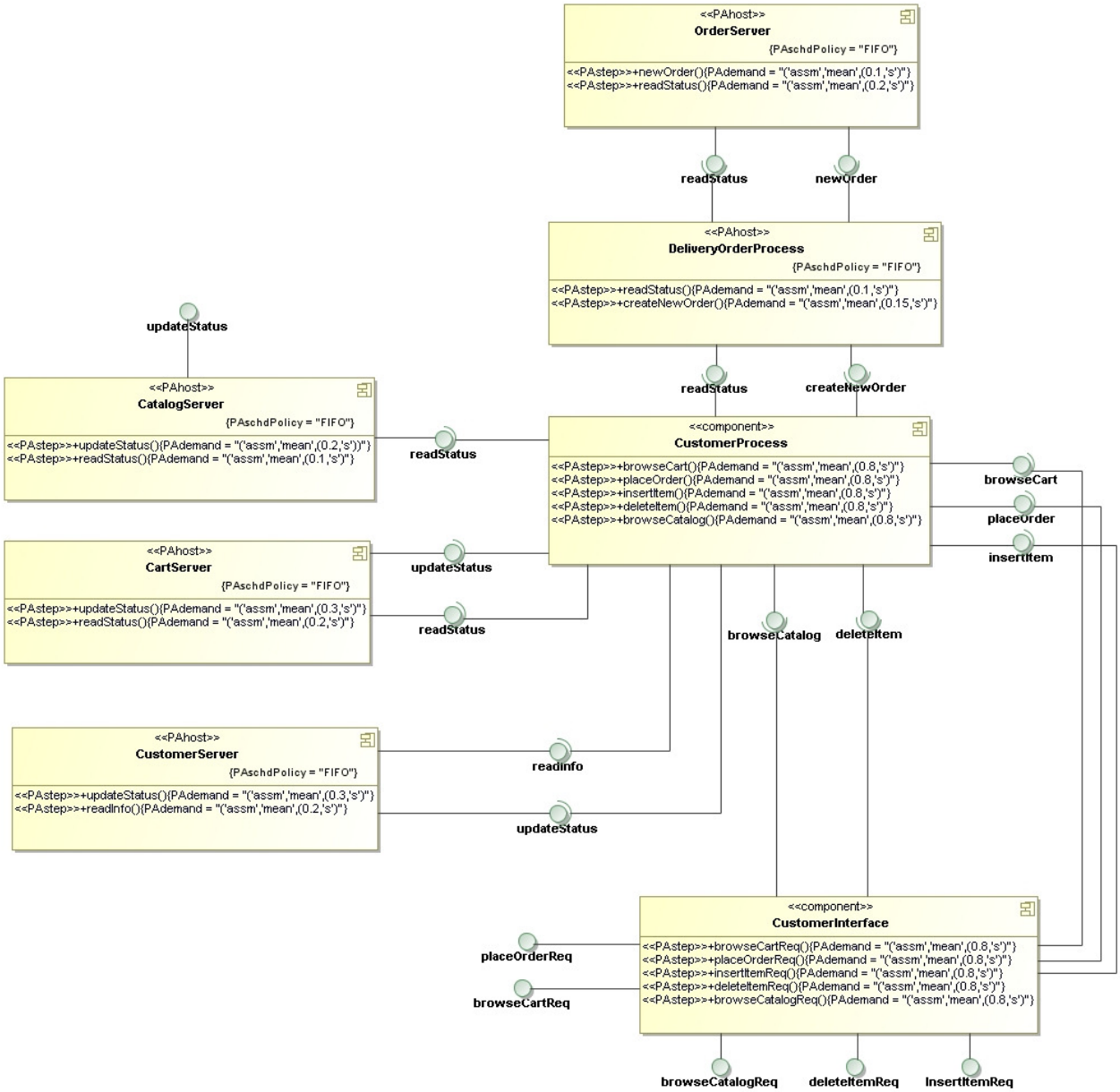


Figure 10: Component Diagram

The considered (sub)system is composed by six components. Four of them (OrderServer, CatalogServer, CartServer, CustomerServer) interact with the corresponding databases by easying the insertion, deletion, reading and updating of data. These components allow asynchronous communications, hence all the interfaces they provide contain asynchronous operations. The remaining three components, namely CustomerInterface, CustomerProcess and DeliveryOrderProcess, provide the GUI to interact with the system, the back end process that manages all her/his request (as made through the GUI) and the business process to manage the order delivery, respectively. These components interact asynchronously with the DB servers and synchronously among them. Thus the interfaces they provide respect such properties.
Finally, since the component diagram in the figure only describes the portion of the e-commerce system considered, it does not contains the components and the interfaces that are not involved in the use cases considered. For example the component that allows the Supplier to delete or insert an item in the catalogue is missing, as well as the corresponding CatalogServer provided interfaces that manage such operations on the corresponding database.

CustomerProcess and CustomerInterface are components running at the user side. There are instances of them dedicated to each user logged in the system. This design choice makes the CustomerProcess and CustomerInterface components unshared resources (i.e. passive components) and hence their execution times constitute delays.

6.3.1 Sap•One annotations

The Component Diagram (CD) is annotated with parameters related to the components, as **each component is transformed into a Queueing Network server center**. Example paramters are: the kind of center (i.e. waiting or delay), and the scheduling policy that it uses to extract requests from its queue (e.g. FIFO, IS etc).

Stereotype Syntax	Apply Stereotype on (UML metaclasses)	Semantic
<<PAhost>> {PAschdPolicy='type'}	On the uml.**Component**	If a component is extended with this stereotype its queue has a *type.* If a component is not extended, then it corresponds to a delay resource without queue and with an infinite number of instances.
<<PAstep>> { PAdemand = ('assm','mean',(*value,'unit* time'))} XOR { PAdelay = ('assm','mean',(*value,'unit* time'))}	On the component uml.**Operation** implementing the homonymous provided uml.**Interface**.	The execution time to satisfy each services request. PAdemand must to be used for active components whereas PAdelay must be used for passive ones. PAdemand and PAdelay tags are mutually exclusive because a component can't be active and passive at the same time.

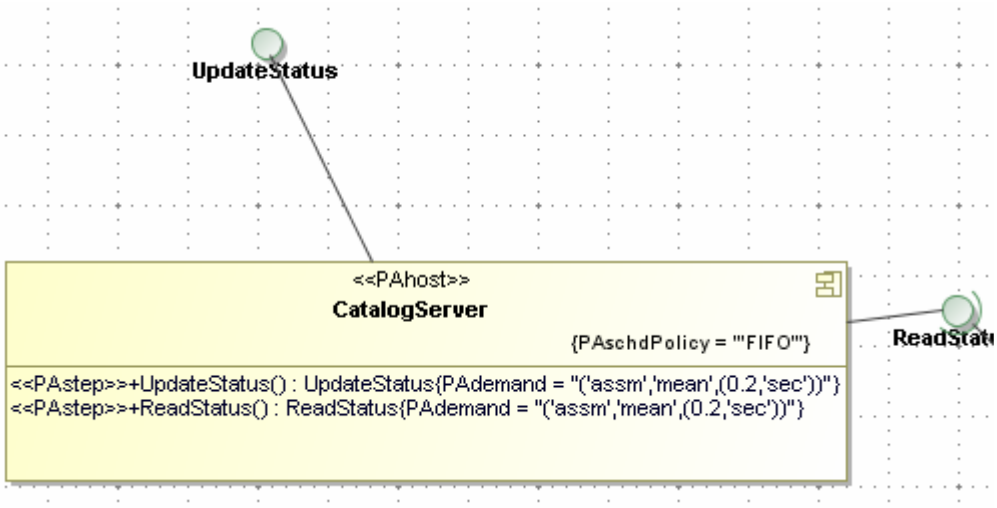
Table 2: Sap•One Stereotypes used on CD.

Each software component is annotated with the **<<PAhost>> stereotype** , and the methodology works under the hypothesis that every component is allocated on a different active resource.
The tag value *PAschdPolicy* denotes the scheduling policy of the component queue.
The **«PAstep» stereotype** is associated to the component operation implementing the homonymous interface: both tag values *PAdemand* (for active components) and *PAdelay* (for passive components) model the component execution time to satisfy a request exposed by the interface.
In particular the syntax to use within the CD is summarized in Table 2.

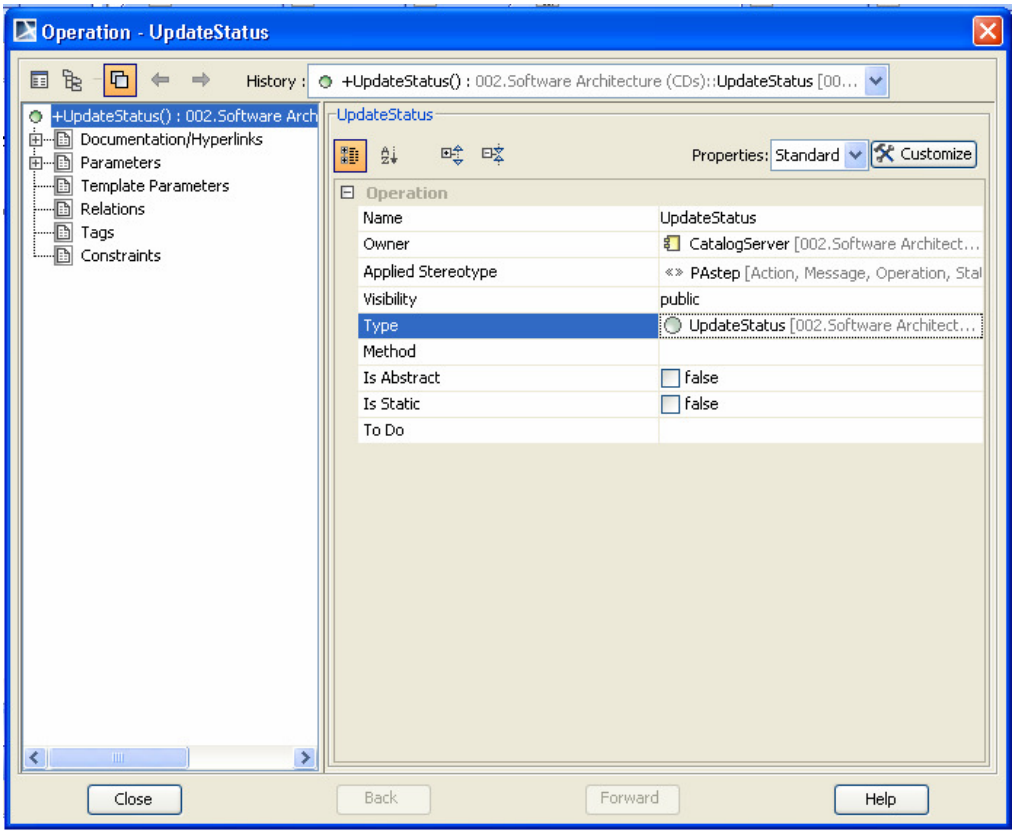
6.3.1.1 Assumptions

Likewise to the previous versions of MOSQUITO, the input model for building a QN platform independent has to be made of a Component Diagram and a set of Sequence Diagrams, under the following constraints:

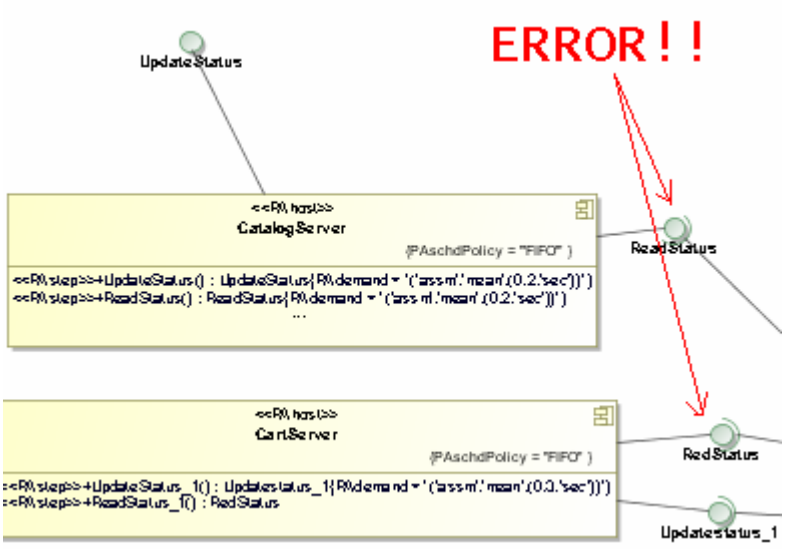
1) Each component must have the following structure:



In particular, for each provided interface you must specify a component operation that returns the interface itself. The screenshot here below show this setting for an UpdateStatus operation.



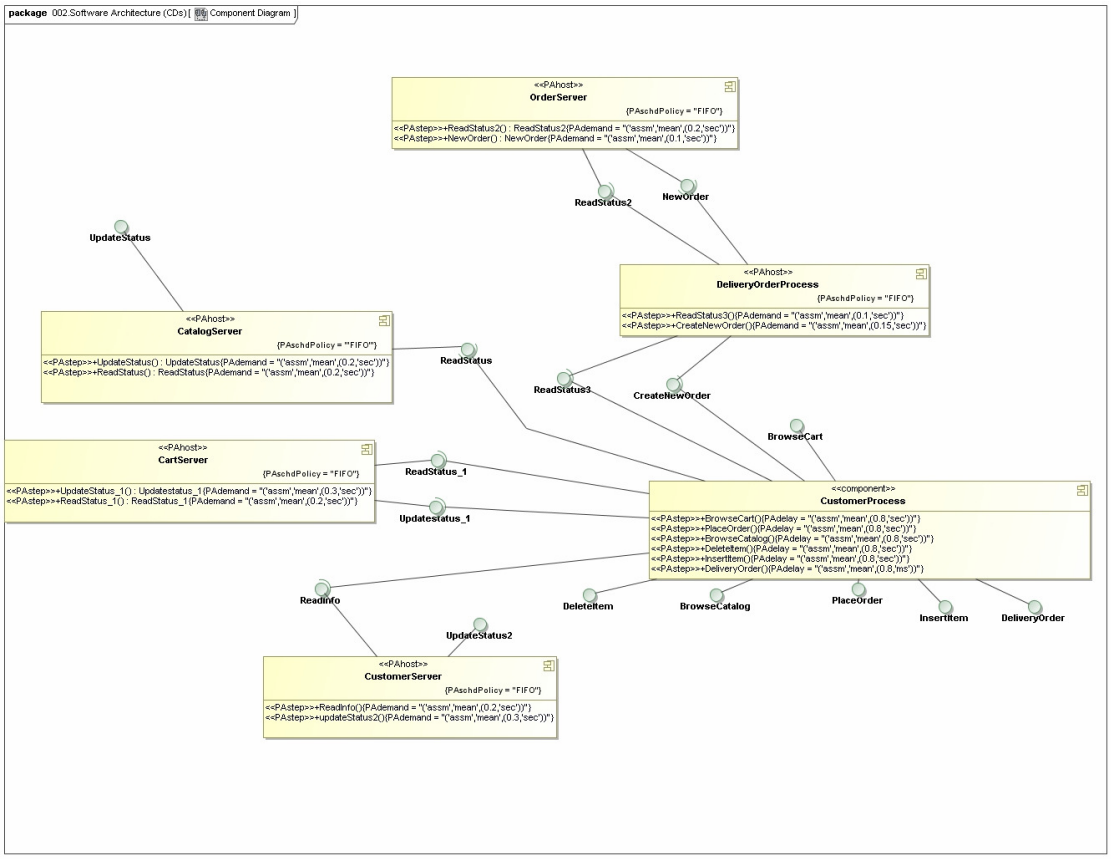
2) It is not possible to use the same interface in more than one component. This can cause a semantic error during the transformation.



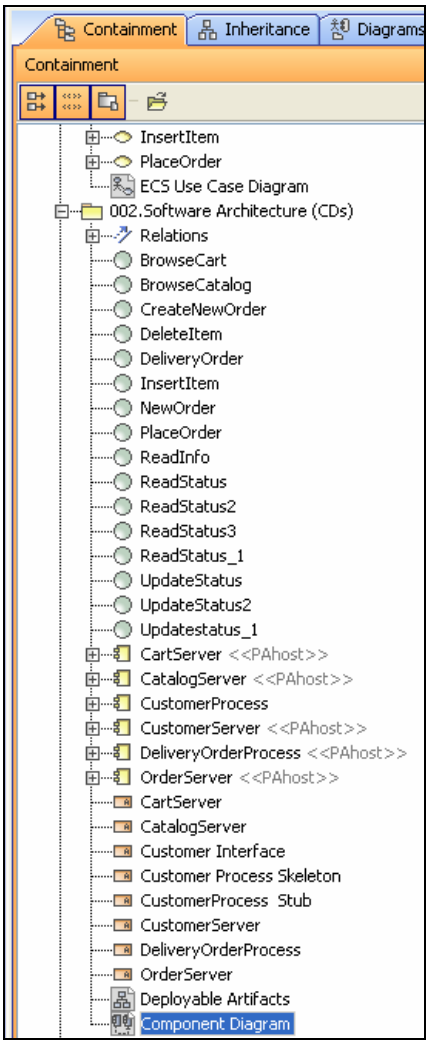
3) If the component is an active one then it must have a PAhost stereotype with the scheduling policy field initialized. The possible values that this field can assume are FCFS, IS, PS, BAS.

4) Every operation must have a PAstep stereotype with PAdemand or PAdelay tag initialized, respectively, if the component is an active or a passive one.

Here below we can see an example of Component Diagram created with MagicDraw that is MOSQUITO1.5 compliant.



Note that the exporting of MagicDraw models is not performed starting from the picture that you see in the main editor of Magic Draw, but starting from the UML2 tree on the left side of the tool screenshot (shown in the next figure). You must be sure that the diagram is aligned with the tree so you are sure that what do you see in the graphics is what MOSQUITO will process.



6.3.2 Prima-UML annotations

No Prima-UML annotations are required on the Component Diagram.

6.4 Sequence Diagram

Interaction diagrams are a common mechanism for describing systems, at varying levels of detail, in a way comprehensible by both software designers and potential end users and stakeholders of the system. Typically the interaction diagrams do not describe the whole system behavior but selected execution traces. There are normally other legal and possible interactions that are not contained within the drawn diagrams. Originally, **interaction diagrams represent system objects and how they interact**. In the new conception (UML 2.0), such diagrams describe the system by means of participants (that can be system modules at a different level of abstraction such as subsystem or component instances, objects and so on) and how they interacts each other to accomplish a task. In the UML terminology, interactions are units of behavior that focus on the observable exchange of information between elements (such as objects) in form of messages. UML 2.0 defines four different interaction diagrams: sequence diagram, communication diagram, interaction overview

diagram and timing diagram. Among these diagrams both **Sap•One and PRIMAUML methodologies** use Sequence Diagrams: they specify **messages that are exchanged between component instances** (whose types are depicted on Component Diagram) that are in charge to provide the system functionalities (i.e. the use cases depicted on Use Case Diagram).
As example scenario the **PlaceOrder sequence diagram** is presented in Figure 11. The Sequence Diagram is composed by several (nested) combined fragments and their operands. In particular, it shows an example of two alternative behaviors specified by means of an alt operator with two InteractionOperand. A Customer asks for an order placing through his own CustomerInterface and, first of all, the CustomerProcess reads the cart status on the CartServer. If the cart is empty the order cannot be placed (see the second interaction operand in the alt combined fragment). Otherwise the CustomerProcess proceeds in collecting the customer information (such as for example his mail address) and creates a new order in the Order DB. Finally it empties the customer cart. In this scenario it is assumed that all the information the customer must provide to place an order is collected before and the PlaceOrder request corresponds to the final submission from the customer. Moreover, the payment procedure is encompassed in a different use case. For example, if the customer decided to pay by a credit card, the Payment&Invoice use case (see Figure 9) will be in charge to manage the bank transfer.

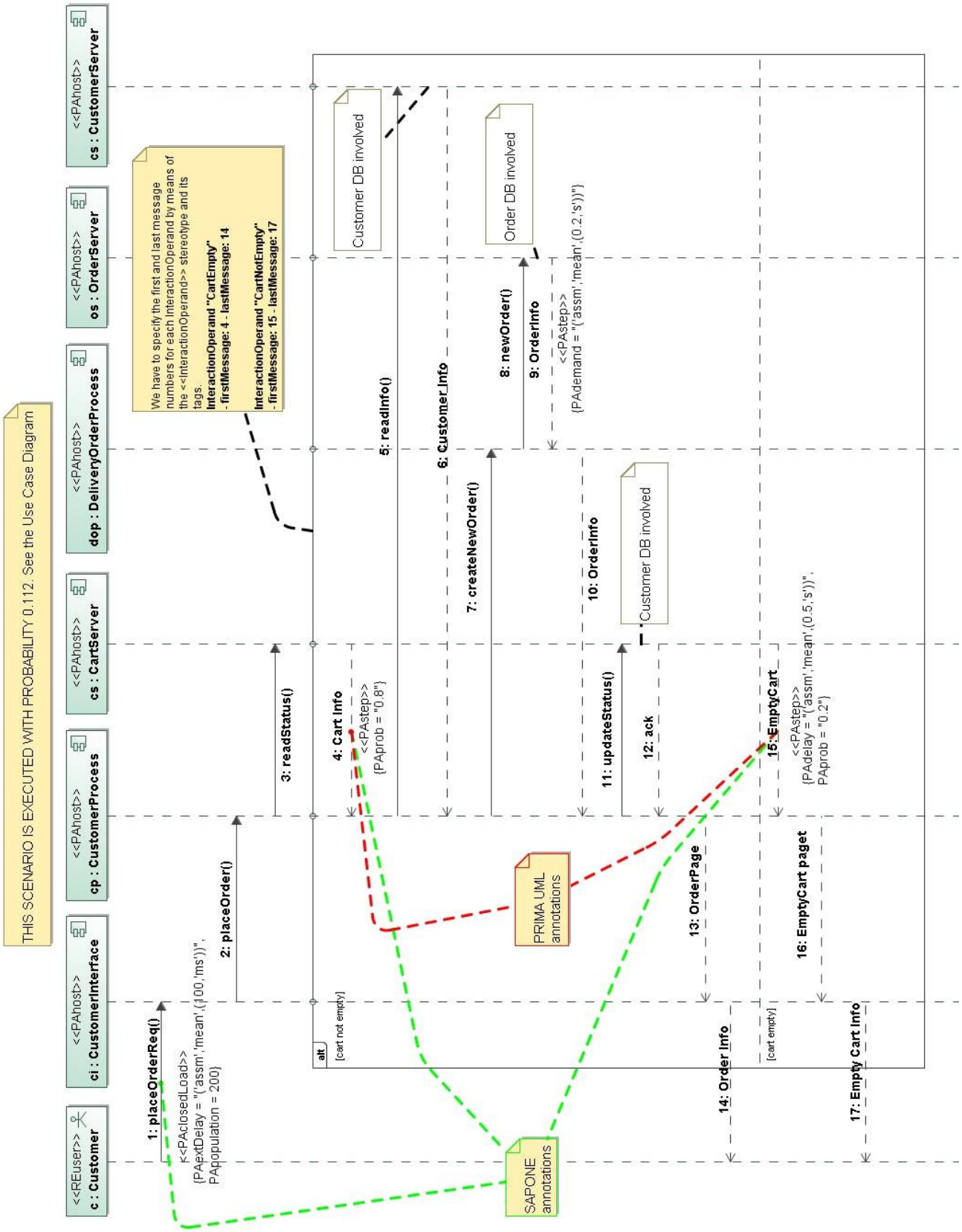


Figure 11: the Sequence Diagram

6.4.1 Sap˙One annotations

The **Sequence Diagram (SD)** must be enriched with the information in the following table.

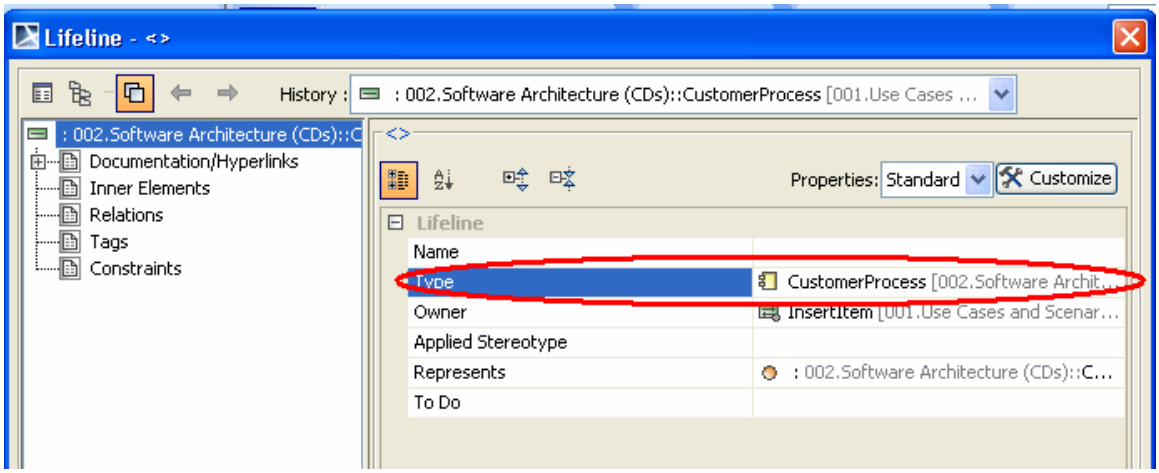
Stereotype Syntax	Apply Stereotype on (UML metaclasses)	Semantic
<<PAclosedLoad>> {PApopulation=population quantity, PAextDelay=('assm', 'mean' ,(30,'ms'))}	On the first uml.**Message** of the Sequence Diagram	This stereotype denotes the closed workload of the scenario depicted on the Sequence Diagram, in particular its population and the think time.
<<PAstep>> {PAprob = value}	On the first uml.**Message** used on the two uml.**InteractionOperand** (i.e. *if* and *then* operands)of the **Alt** uml.**CombinedFragment**	It denotes the probability to activate the message in the uml.**CombinedFragment**. With probability 1-value the control is given to the first message following the alt compartment.

Table 3: Sap•One Stereotypes used on SD.

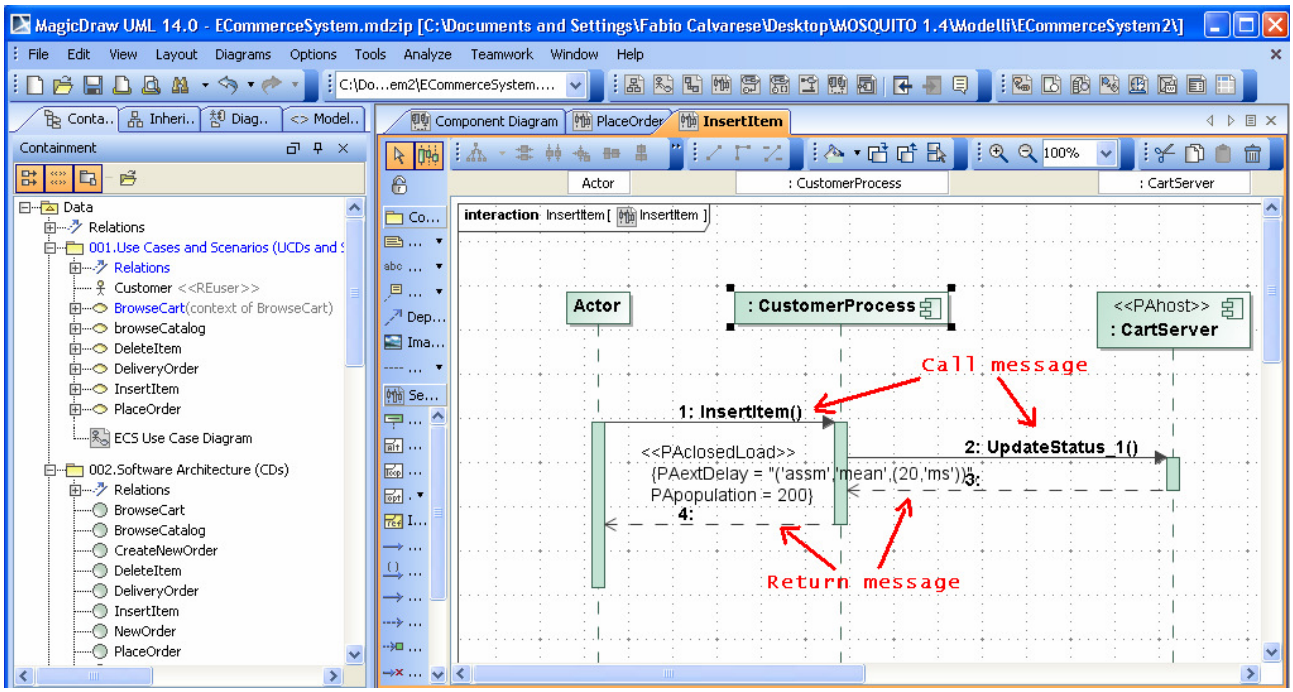
The **stereotype «PAclosedLoad»** specifies the closed workload intensity and it is used to annotate the first message of every Sequence Diagram, where the used tag values are: *PApopulation* (i.e. number of user) and *PAextDelay* (i.e. required time to execute a request). Possible alternative behaviors in the UML interaction fragments are annotated with the **<<PAstep>> stereotype.** The *PAprob* tag value specifies the execution probability of each behavior.

6.4.1.1 Assumptions

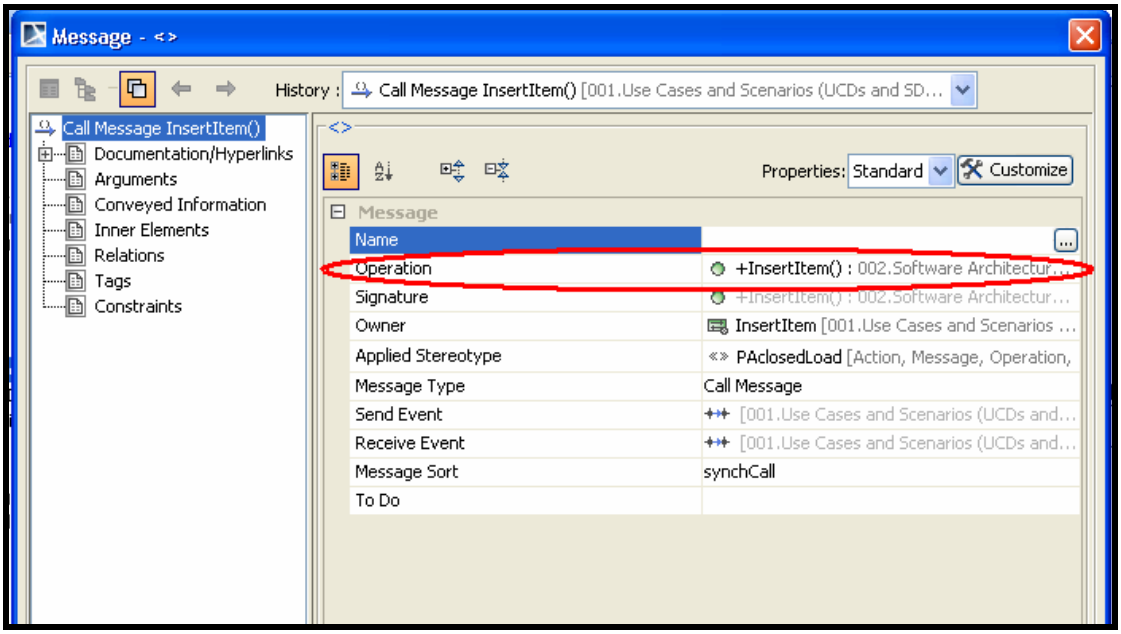
1) Every Instance of the Sequence Diagram (with the exception of the actor) must be bound to the associated component as it is shown in the next picture. This is necessary to bind the behavioural system specification at the structural one.



2) Every message in the Sequence diagram must be a "Call message". For every created Call Message you must specify a Return message. This entails that the number of Call Messages in the diagram is always the same as the Return Messages.



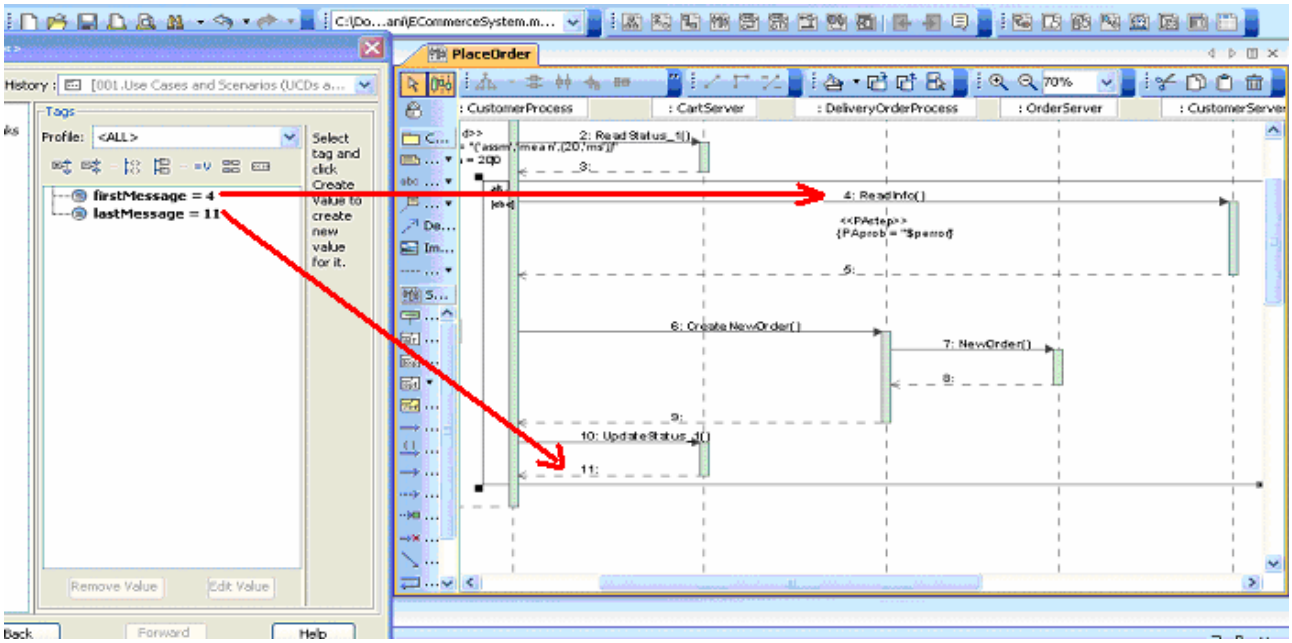
3) Every Call Message must have the property "operation" initialized with the name of the operation to invoke in the receiving component.
Next picture show this setting for the CallMessage that invokes the InsertItem operation.



4) The first CallMessage of each Sequence Diagram must be stereotyped with the <<PAClosedLoad>> stereotype.

5) The first message entering in each operand of an alternative fragment must be stereotyped with the <<PAstep>> stereotype using the PAprob tag. This tag indicates the probability to invoke all the Call Messages into the fragment operand. This value must fall between zero and one.

6) For every used fragment in the sequence diagrams you must specify the first and the last message id properly contained in the fragment. This operation must be performed using the firstMessage and lastMessage tags of the stereotype Interaction Operand, as showed in the example here below.



6.4.2 Prima-UML annotations

The Sequence Diagram must be enriched with the information in the following table.

Stereotype Syntax	Apply Stereotype on (UML metaclasses)	Semantic
<<PAstep>> {PAprob = value}	On the first uml.**Message** used on the two uml.**InteractionOperand** (i.e. *if* and *then* operands)of the **Alt** uml.**CombinedFragment**	It denotes the probability to activate the message in the uml.**CombinedFragment**. With probability 1-value the control is given to the first message following the alt compartment.
<<PAstep>> {PArepetitionFactor = value}	On the first uml.**Message** exchanged in the **Loop** uml.CombinedFragment	It denotes the number of the iterations in the loop fragment.

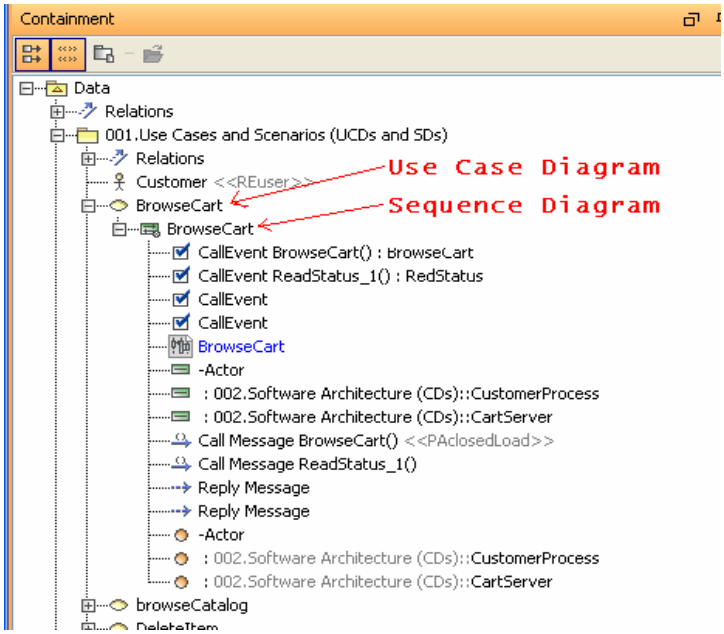
Table 4: Prima-UML Stereotypes used on SD.

It is worth to note that the <<PAstep>> stereotype is applied on the first message of the Alt uml.CombinedFragment in the same manner as the Sap•One methodology.

6.4.2.1 Assumptions

The only difference with the SAPone methodology is described here below:

1) You must specify a Sequence Diagram for each use case in the Use Case Diagram. In order to associate the Sequence Diagram at the Use Case, you must create it as a child in the Uml2 tree hierarchy, as showed in the example here below.



6.5 Deployment Diagram

A **Deployment Diagram** shows the configuration of run-time processing elements and the software components that live on them. It is a graph where nodes (i.e., the processing elements, uml.Node) are connected by communication path (i.e. uml.CommunicationPath). **Nodes can contain artifacts (uml.Artifact) that manifest components** (see Figure 12). Therefore a DD shows the mapping of components on processing elements (see Figure 13).

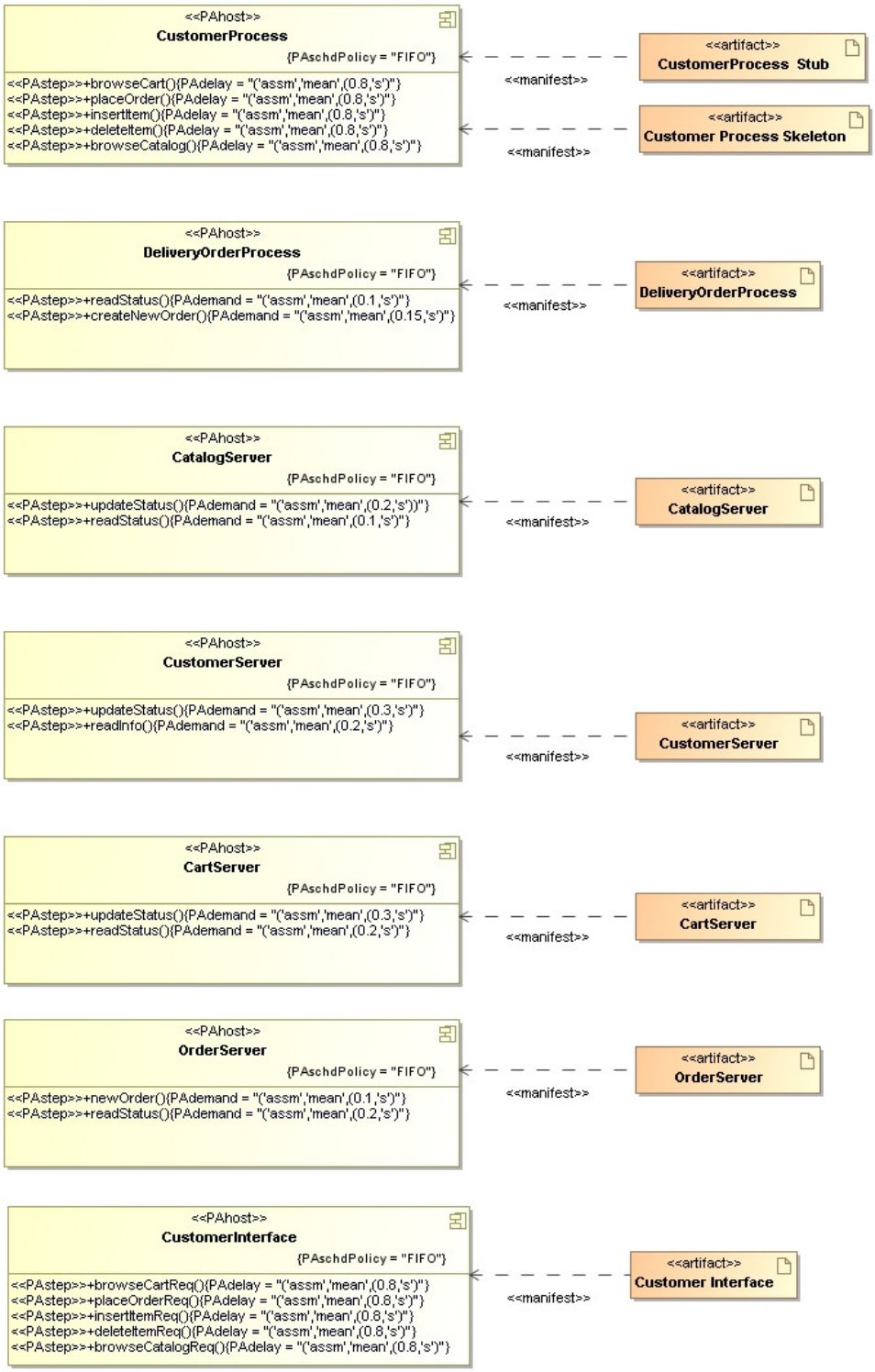


Figure 12: The artifacts that manifest each component

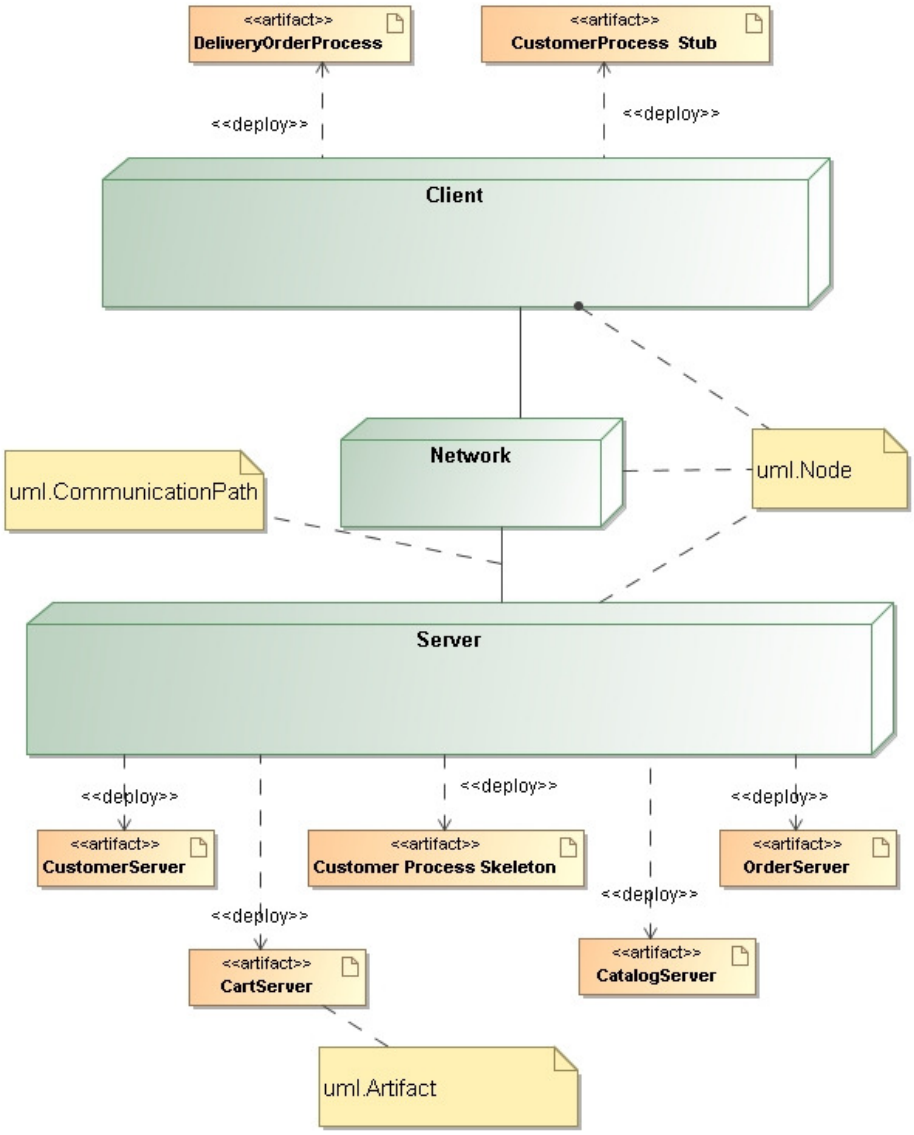


Figure 13: Deployment Diagram

Figure 12 shows the uml.Artifacts that manifests uml.Components (see the <<manifest>> stereotyped dependency between the uml.Component and its own uml.Artifacts), whereas Figure 13 shows uml.Artifacts deployed on uml.Nodes.

6.5.1 Sap•One annotations

No Sap•One annotations are required on the Deployment Diagram.

6.5.2 Prima-UML annotations

The QNM topology is derived from the annotated DD: the interacting component instances depicted on the SDs (as uml.Lifelines) are deployed (by means of uml.Artifacts, see Figure 12) on a node equipped with a suitable set of hardware resources. Such hardware resources are shown on the deployment diagram in Figure 14.

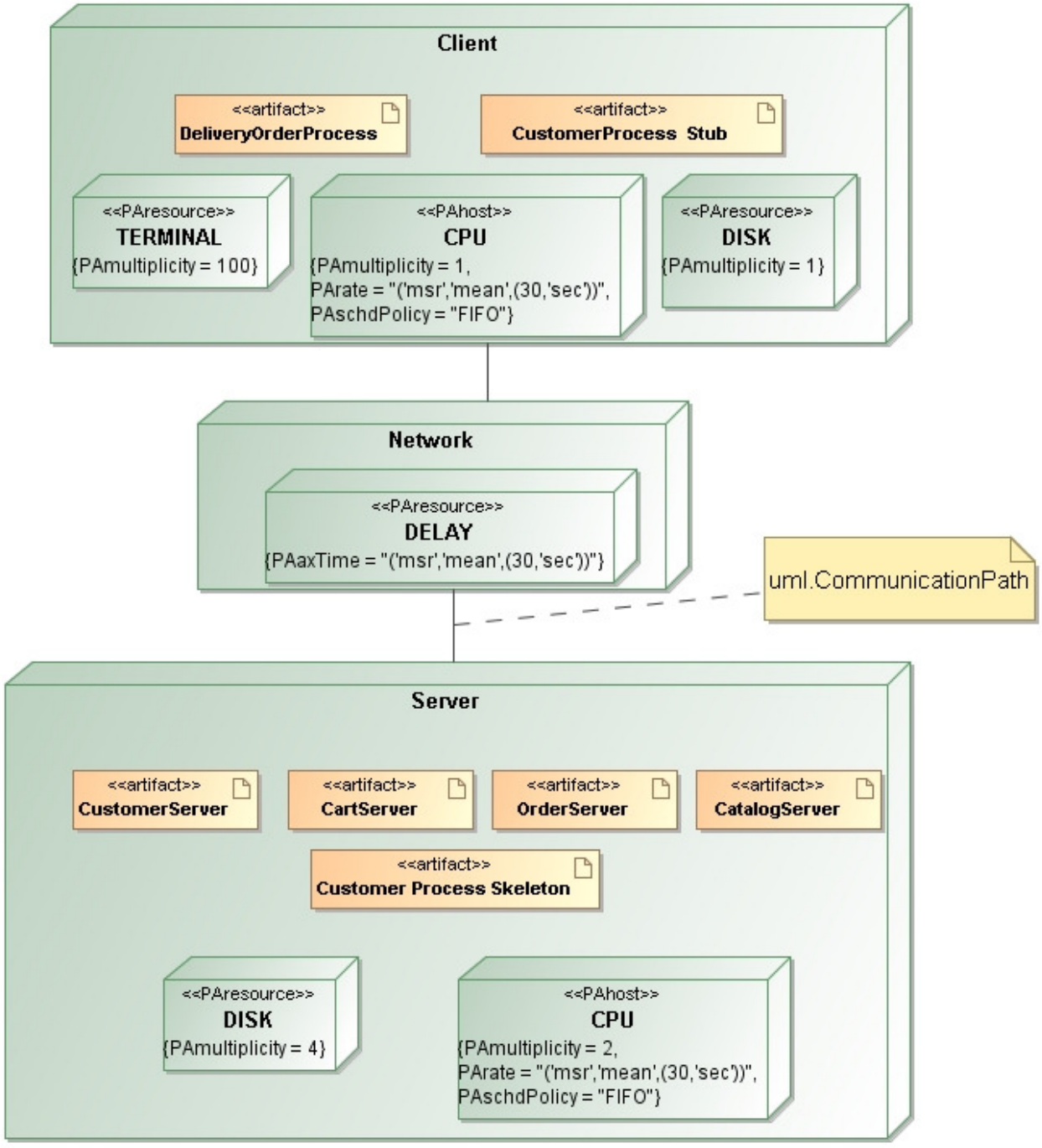


Figure 14: Annotated Deployment Diagram for the Prima-UML methodology.

Hardware resources are distinguished in active and passive ones by means the <<PAhost>> and <<PAresource>> stereotypes, respectively [3]. The Prima-UML methodology makes use of three different kind of resources over a uml.Node:

- CPU, as the name itself suggests, is a (active <<PAhost>>) resource representing CPUs;
- DISK that represents a storage (passive[4] <<PAresource>>) resource;
- TERMINAL[5] that represents users/jobs that access the container node.

[4] A **processing resource** (i.e. PAhost) is a device which has processing steps allocated to it. A **passive resource** (i.e. PAresource) represents a resource protected by an access mechanism (e.g., a semaphore), which is accessed during the execution of an operation.

[5] CPU, DISK and TERMINAL are reserved words.

Table 5 specifies the additional information that has to be specified for such kind of resources.

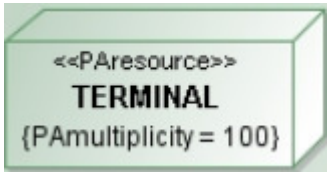
Stereotype Syntax	Apply Stereotype on (UML metaclasses)	Semantic
<<PAhost>> **CPU** {PAschdPolicy = value, PArate = value, PAmultiplicity = value}	On **uml.Node**.	A **processing resource** is a device which has processing steps allocated to it. *PAmultiplicity* defines how many instances of the CPU resource are available on the containing uml.Node.
<<PAresource>> **DISK** {PAmultiplicity = value}	On **uml.Node**.	A **passive resource** represents a resource protected by an access mechanism (e.g., a semaphore), which is accessed during the execution of an operation. *PAmultiplicity* defines how many instances of the DISK resource are available on the containing uml.Node.
<<PAresource>> **TERMINAL** {PAmultiplicity = value}	On **uml.Node**.	It represents users/jobs that access the containing node. *PAmultiplicity* defines how many users/jobs access the containing uml.Node.
<<PAresource>> **DELAY** {PAaxTime = value}	On **uml.Node**.	It represents a network (i.e. LAN, WAN) connecting two Node. *PAaxTime* specifies the delay introduced by the network communication.

Table 5: Prima-UML Stereotypes used on DD.

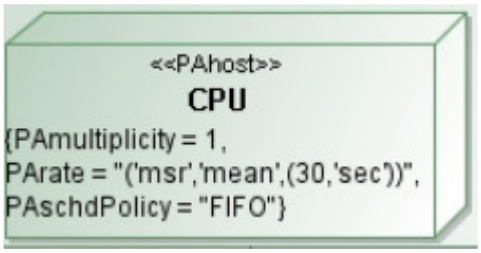
6.5.2.1 Assumptions

The hardware resources supported from the methodology are: the CPU, the DISK, the TERMINAL and a DELAY resource representing network connections. The assumptions on the input model are as follows:

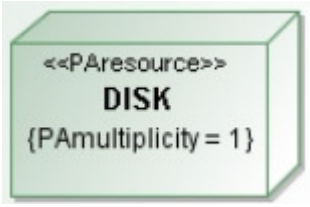
1) Only one Terminal resource must appear in the model. A Terminal resource must be specified as a node named “TERMINAL”. This kind of node must be stereotyped with a <<PAresource>> stereotype that indicates the multiplicity of the hw resource. It must be allocated in a deploy node where at least the CPU resource is allocated. Here below you can find an example.



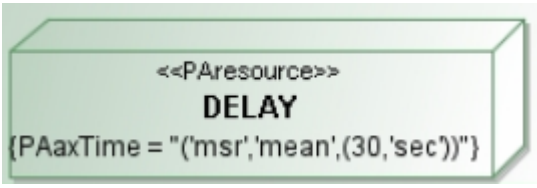
2) The CPU resource must be specified as a node named "CPU". This kind of resource is an active resource and it must be stereotyped with the <<PAhost>> stereotype to indicate the multiplicity, the rate and the scheduling policy used from this resource. In a model can be used more than one CPU in different deploy nodes. The CPU resource is displayed here below.



3) The DISK resource must be specified as a node named "DISK". This kind of resources can be active or passive, so it can be stereotyped with <<PAhost>> or <<PAresource>>. It can be used in a deploy node where CPU and TERMINAL resources exist. Here below a DISK resource.



4) The DELAY resource can be specified as a node however named (for example LAN, WAN , DELAY), and it represents a resource that introduces a delay in the communication between nodes of the Deployment Diagram. It must be allocated in a node with no other resource, and it must be stereotyped with the << PAresource >> stereotype, as showed in the picture here below.



7 References

[1] A. Di Marco, P. Inverardi *"Compositional generation of Software Architecture Performance QN Models"*. Proc. 4th Working IEEE/IFIP Conference on Software Architecture (WICSA'04), *2004.*

[2] V. Cortellessa, R.Mirandola *"PRIMA-UML: a Performance Validation Incremental Methodology on Early UML Diagrams"*, Science of Computer Programming , 44 (2002), no.1, pagg.101-129.

[3] OMG, *UML Profile, for Schedulability, Performance, and Time*, Version 1.1, formal/05-01-02.

[4] C. Smith, Performance Solutions, Addison Wesley, 2002.

8 Appendix

In this appendix we report details of the methodology mechanisms.

8.1 Queuing Network process creation (SAP·one methodology)

The creation process of the queuing network through the Sap·one methodology can be summarized in two steps:

- *Identification of the service centers and their carachteristics:* A delay or waiting service center is associated to every component of the Component Diagram. An offered service that corresponds, in practice, to every interface of the component, indicates what job classes are managed by each server.
  Service center characteristics (e.g. time service, kind of center and scheduling policy) are defined from the associated tag values.

- *Classes chain and workload definition*: The Sequence Diagrams (with the annotated information) represent the workload intensity, while the net topology is defined analysing the dynamics of communication between components. The idea is to associate a connection in the queuing network from each sender component (i.e. a component that sends a messages in some Sequence Diagrams) to each receiving component. The receiving component will have infinite queue if the interaction is asynchronous, whereas a null queue if synchronous.

SAP·one defines a set of translation rules combined to various architectural patterns corresponding to at interaction fragment of UML Sequence Diagrams. In this way it is possible to generate the complete architectural topology stepwise while parsing the UML diagrams [1].

8.2 Execution Graph creation (Prima-UML methodology)

The translation rules used from the Prima-UML methodology can be summarized in the following steps.

Estimation of scenarios execution probability:
In Figure 15 an annotated UCD is shown: users and arcs are annotated with weights that lead to compute the probability that each use case is expected to occur. Note that, early in the lifecycle, these weights come out from designer's guesses on the operational profile, whereas later they can be obtained from actual measures.

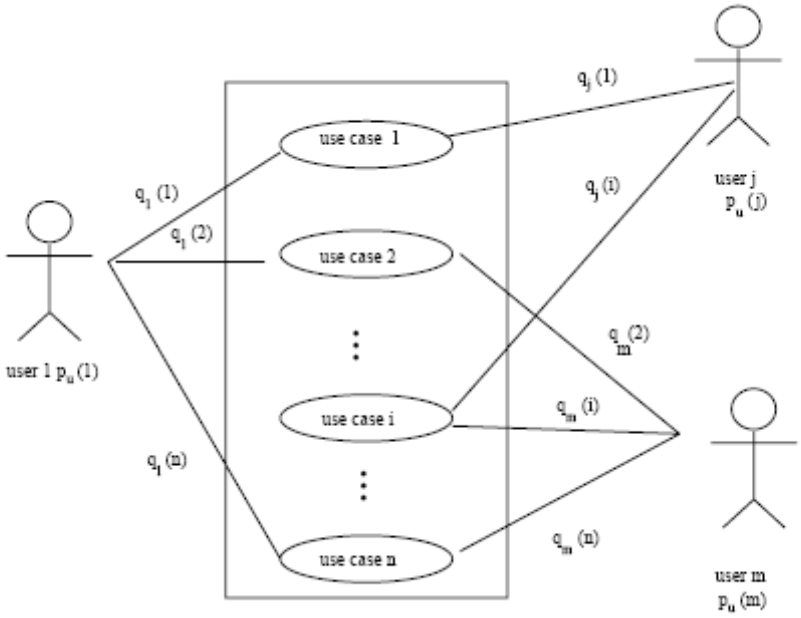


Figure 15: Generically annotated UCD

Let us suppose to have a UCD with *m* types of users and n use cases. Let $p_u(i)$ (i=1,…,m) be the *i*th user type probability of usage of the software system (i.e. <<*REuser*>> *REaccessprob*) . We have that:

$$\sum_{i=1}^{m} p_u(i) = 1$$

Let $q_i(j)$ (j=1,…,n) be the probability that the *i*th user makes use of the software system by executing the use case *j* (i.e. <<*REserviceUsage*>> *REserviceprob*). We have that:

$$\sum_{j=1}^{n} q_i(j) = 1$$

The probability for whatever scenario *j* (described by means of a Sequence Diagram (SD) associated to the UseCase *j*) to be executed is then given by

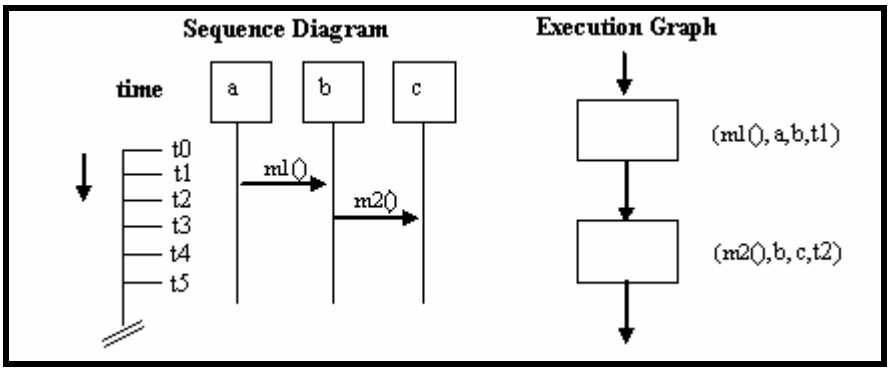
$$P(j) = \sum_{i=1}^{m} p_u(i) q_i(j) = 1$$

This can be equivalently expressed as:

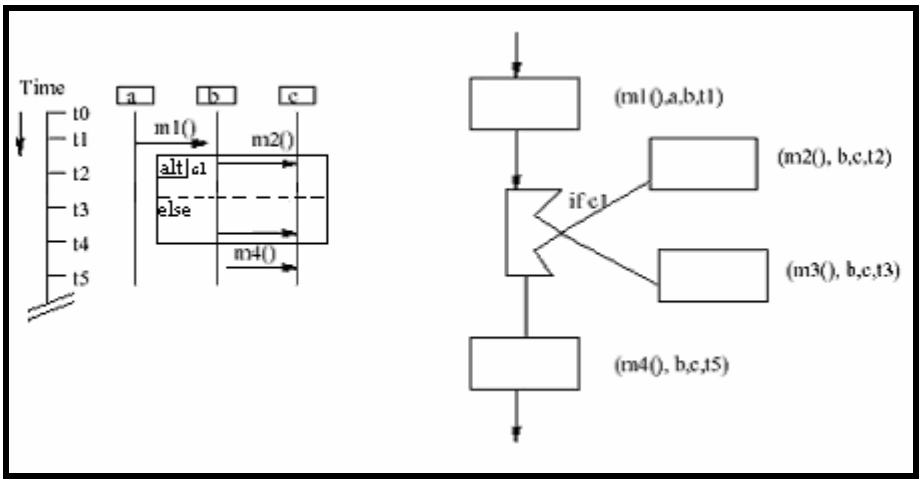
$$REprob_{REservice_j} = \sum_{i=1}^{m} REaccessprob_{REuser_i} \cdot REserviceprob_{REuser_i\ REservice_j} = 1$$

Sequence Diagram processing and meta-Eg building:

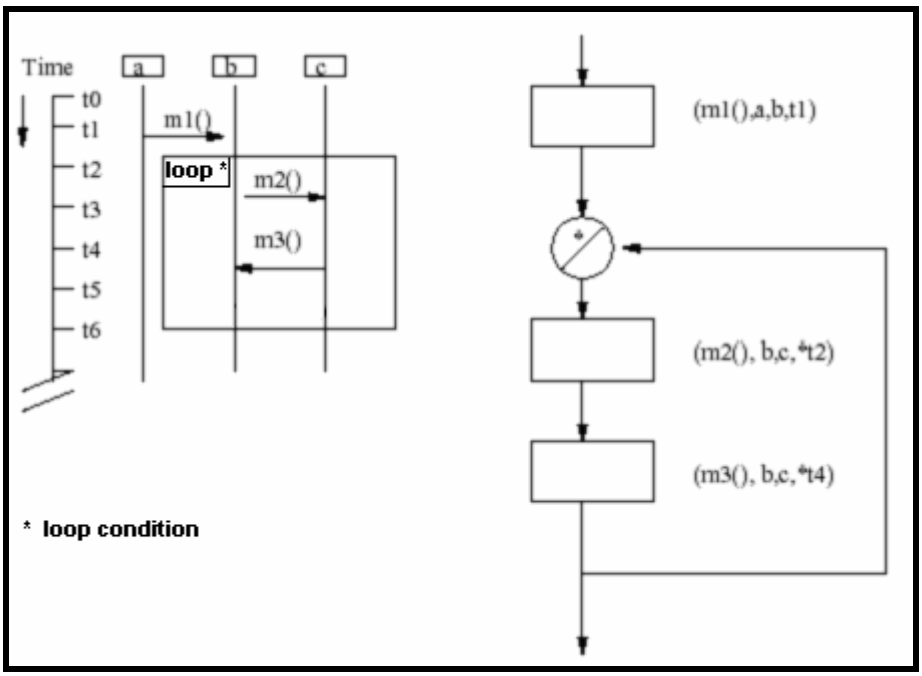
In this step an algorithm is used to transform all the Sequence Diagrams in an Execution Graph that is called a meta-EG [2]. The meta-EG is a platform independent annotated Execution Graph. The idea of the algorithm it is to apply the following translating rules for Sequence Diagram patterns.



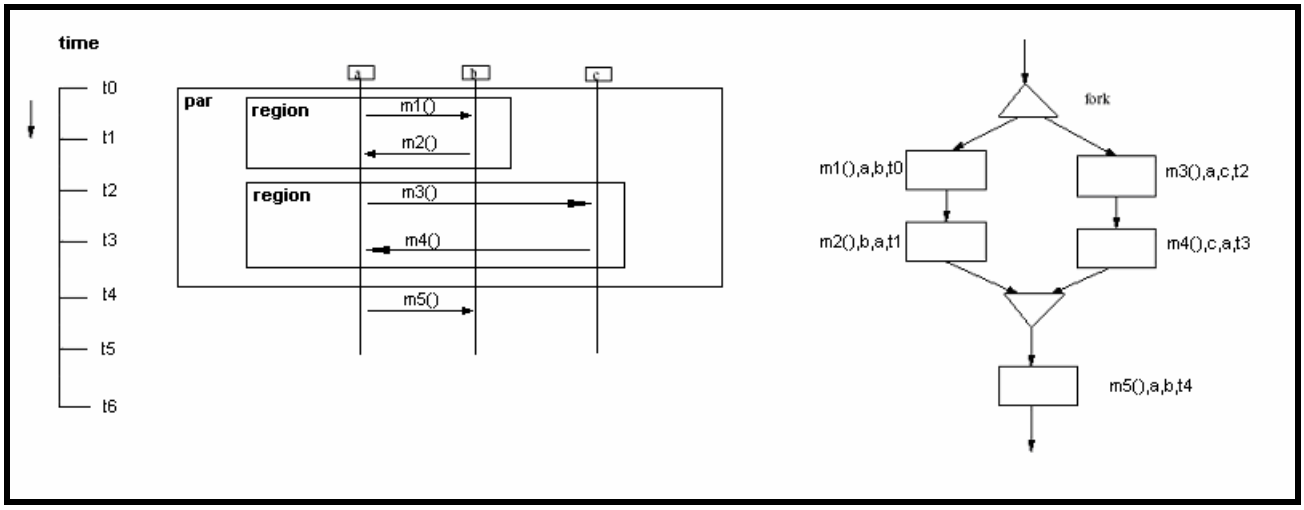
Translation rule for the sequential steps.



Translation rules for the alternative fragmetn.



Translation rule for the loop fragments.



Translation rule for the parallel fragments.

For what concerns Sequence Diagram fragments, the following mapping is adopted.
The Reference operator, corresponding at the key word R*ef,* denotes that a Sequence Diagram includes a defined behaviour in another Sequence Diagram. This is translated so that the queuing network corresponding to the first diagram merges the sub-net corresponding to the second diagram.
The alternative operator, corresponding at the key word *Alt*, denotes the presence, in the diagram, of a choice between two behaviours, or more simply the choice to execute or not a specific behaviour. The methodology translates the operator as a net branching point with probabilities associated to the matching patterns.
The Parallel operator, corresponding at the key word *Par,* denotes two or more different patterns to be executed in parallel. It is translated with a pair of fork/join nodes in the network.
The methodology is able also to translate other operators, as described in [1],that are not yet implemented in MOSQUITO.